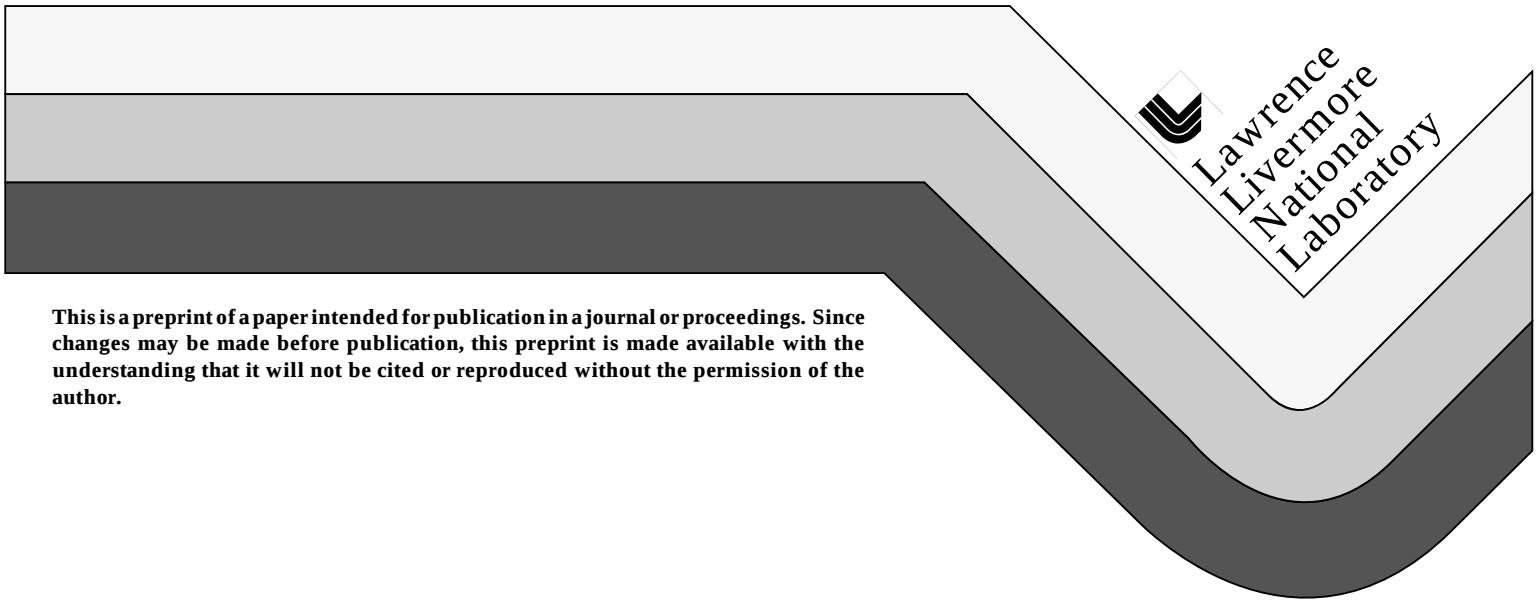
UCRL-JC-119754
PREPRINT

Class of Service in the High Performance Storage System

S. Louis
D. Teaff

This paper was prepared for submittal to the
IFIP International Conference on Open Distributed Processing
Brisbane, Australia
February 21-24, 1995

January 10, 1995



Class of Service in the High Performance Storage System

S. Louis[a] and D. Teaff[b]

[a]Lawrence Livermore National Laboratory, Livermore CA 94551-9900 USA louis@nersc.gov

[b]IBM U.S. Federal, Houston TX 77058-1199 USA teaff@vnet.ibm.com

Quality of service capabilities are commonly deployed in archival mass storage systems as one or more client-specified parameters to influence physical location of data in multi-level device hierarchies for performance or cost reasons. The capabilities of new high-performance storage architectures and the needs of data-intensive applications require better quality of service models for modern storage systems. HPSS, a new distributed, high-performance, scalable storage system, uses a Class of Service (COS) structure to influence system behavior. We summarize the design objectives and functionality of HPSS and describe how COS defines a set of performance, media, and usage attributes assigned to storage objects managed by HPSS servers. COS definitions are used to motivate appropriate behavior and service levels as requested (or demanded) by storage system clients. We compare the HPSS COS approach with other quality of service concepts and discuss alignment possibilities.

Keyword Codes: C.4; H.3.4
Keywords: Performance of Systems; Information Storage and Retrieval, Systems and Software

1. INTRODUCTION

A mass storage system is that portion of a computing facility responsible for long-term storage of information. These systems are shared among users and organized around specialized hardware devices. The complexity of storage systems has undergone rapid advancement over the past twenty years as modern computers placed increasing demands on support services. The evolution of storage architectures has been shaped by ever-larger capacities and the rapid growth of interactive processing, networks, and distributed computing. Storage systems grew from simple, large peripheral disk and tape devices and utilities, through centralized, but shared service nodes, and finally to large, complex, often highly distributed, systems supporting powerful supercomputers and parallel processors.

The sheer size of some storage problems meant that the largest systems were developed at organizations such as large government research laboratories and scientific supercomputer centers, using in-house systems engineering expertise. These individual efforts brought about systems that were heavily dependent on unique elements at each site. Unfortunately, developers usually made assumptions about who users were and how storage capabilities would be used, forcing users to interact in prescribed ways to use archival services. Varying levels of transparency were provided to reduce the complexity of system interaction, but the disadvantage of some transparencies is that efficiency may be lost in resource utilization or performance. Different levels of service quality were generally not offered. While developers of early storage systems were certainly aware of service level issues, the term *quality of service* (QoS) sometimes became a

catch-all bucket into which were deposited all manner of long-term, difficult implementation issues concerning successful administration and operation of a high-performance storage system.

Client-server and consumer-provider models have been examined for many years within the mass storage community. The IEEE Mass Storage Systems Reference Models (MSSRMv4 and MSSRMv5) [1,2] identified high-level abstractions that underlie modern storage systems. The IEEE view of a storage system is one or more storage device hierarchies, implemented using an architecture that allows storage services to be distributed throughout the system. Consumers of storage service interact with standardized providers through well-defined Application Programming Interfaces (APIs). These interactions may be subject to several environmental constraints, including *storage system management* policies, administrative requirements, and operational procedures. Storage system management is discussed in MSSRMv5, but QoS is not explored.

During the time that the IEEE MSSRMs were developed, system managed storage initiatives were also launched by the GUIDE and SHARE user groups of IBM equipment. These initiatives stemmed from growing concern about use of manual techniques to allocate and manage large data center storage, and resulted in new technologies associated with the IBM Data Facility Storage Management Subsystem product. Automation functions were applied to the management of storage space, performance, availability, and configuration, but these functions did not address heterogeneous, distributed environments.

In distributed computing efforts, such as the Reference Model for Open Distributed Processing (RM-ODP) [3] and several emerging enterprise management technologies, concepts of QoS, service level agreements, environment rules and contracts play key roles in determining whether users receive services that meet their needs. Although the IEEE storage models are not identical to the RM-ODP storage function, many issues surrounding QoS are common. Alignment of future storage system standards with RM-ODP and QoS may prove beneficial for management of complex, multi-level device hierarchies in highly distributed computing and storage infrastructures. We describe below how HPSS [4,5], a newly developed high-performance storage system, uses a Class of Service (COS) capability allowing users to observe and specify differing service levels within the storage system.

HPSS is a high-performance storage system for highly parallel computers, as well as traditional vector supercomputers and workstation clusters, and is a major development project of the National Storage Laboratory (NSL). The NSL is an industry and U.S. Department of Energy collaborative project organized to investigate, develop, and commercialize new hardware and software technologies for high-performance distributed storage [6]. The principal development partners for HPSS are the U.S. Department of Energy's Lawrence Livermore, Los Alamos, Oak Ridge and Sandia National Laboratories, and IBM U.S. Federal. Other development partners include Cornell University, NASA Lewis, and NASA Langley Research Centers. A major driver for HPSS was to develop a distributed, high-speed storage system that provides scalability to meet demands of new high performance computers and applications where vast amounts of data are generated [7,8], such as those under development in the Department of Energy's Grand Challenges science program, and to also meet the needs of a national information infrastructure [9].

COS in HPSS is not based on the RM-ODP QoS model, but incorporates similar ideas. In RM-ODP, QoS is viewed as a set of *user-perceived* attributes expressed in *consumer-understood* language that describes an available service. A *service-boundary* is defined separating provider and consumer. Consumers see QoS but not necessarily service performance. Similarly, providers see service performance but not necessarily QoS. In contrast, HPSS COS is a set of

system-defined attributes, expressed in a *provider-understood* language that describes storage capabilities. Both QoS and COS help describe the collective behavior of distributed system objects that may be subject to contractual agreements. The COS design attempts to separate consumer requirements from actual storage device characteristics, but COS and related data structures in HPSS are biased toward the service provider. This is because COS was initially implemented to provide single or parallel data transfer capabilities over possibly striped storage devices. Providing a consumer view was of secondary concern during early HPSS design. Enlarging COS beyond its current use to incorporate more of the common QoS parameters (e.g., delay, availability, reliability, accuracy, security) is under consideration.

Extending HPSS COS toward a consumer-oriented view to better serve new non-traditional clients of mass storage suggests aligning future enhancements to COS with RM-ODP standards and QoS. In addition, the RM-ODP Trading Function [10] may also prove useful in implementing *middleware* software solutions to communication, service offer, and service discovery problems between existing storage systems and new types of clients. These problems occur frequently in mass storage applications because services requested from applications do not necessarily coincide with a storage system's internal view of its offered services. Examples of middleware tasks might include deciding which of several replicated copies of data to select based on load or cost optimization schemes, and abilities to screen out or set aside data requests that may result in hundreds or thousands of random tape mounts and tape read requests. Location-independent operations and interfaces may be necessary. Third-party processes to translate consumer-oriented requests to provider-oriented services in a highly distributed storage system would be beneficial.

2. HPSS OVERVIEW AND DESIGN OBJECTIVES

The HPSS software architecture is based on the IEEE MSSRMv5, and is network-centered. The architecture includes a high-speed network for data transfer and a separate network for control (see Figure 1). The control network uses the Open Software Foundation's (OSF) Distributed Computing Environment (DCE) Remote Procedure Call (RPC) technology. In actual implementations, the control and data transfer networks may be physically separate or shared [11]. Another feature of HPSS is its support for both parallel and sequential input/output (I/O) and standard interfaces for communication between processors (parallel or otherwise) and storage devices.

In typical use, clients direct a request to store or retrieve data to an HPSS server. The HPSS server directs the network-attached storage devices to transfer data directly, sequentially or in parallel, to or from the client node(s) through the high-speed data transfer network. Local devices can also transfer data through the HPSS server. HPSS currently supports a TCP/IP socket programming interface and IPI-3 over HIPPI. Future plans include support for Fibre Channel Standard (FCS) and Asynchronous Transfer Mode (ATM) networks. COS specifications can be included with file creation requests to influence behavior of the HPSS servers regarding initial data placement and subsequent data migration operations. The COS identifier becomes part of the persistent HPSS metadata for a new file.

The HPSS I/O architecture is designed to scale as technology improves by using data striping as a parallel I/O mechanism. The system is designed to support application data transfers from hundreds of megabytes up to a gigabyte per second. File size scalability must meet the needs of billions of data sets, each potentially terabytes in size, for total storage capacities in petabytes. The system must also scale geographically to support distributed systems with hierarchies of distinct storage systems. Multiple systems located in different areas must integrate into a single logical system accessible by personal computers, workstations, and supercomputers. HPSS

design was also driven by modularity of software components. Each software component is responsible for a well-defined set of storage objects, and acts as a service provider for those objects.

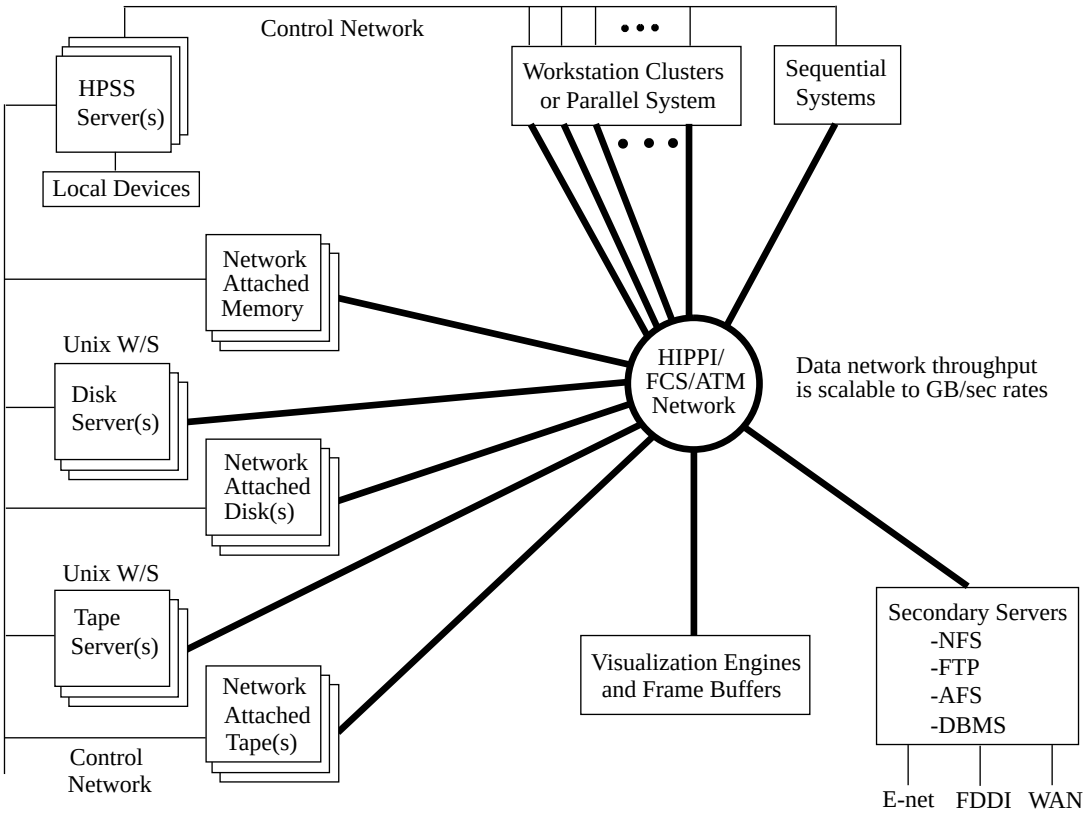


Figure 1. An Example HPSS Configuration

Current applications access HPSS and specify file-related COS characteristics at the file interface level. A COS identifier for a new file can be passed to HPSS using the quote command in FTP or through a Client API call. The Client API also provides an ability to pass prioritized hints that can force the assignment of an appropriate COS for a new file. Files in HPSS are composed of lower-level objects at both a logical and physical level. The management of these lower-level objects and their individual or collective behavior is also controlled through appropriately defined storage class identifiers related to the COS. The importance of new COS capabilities for lower-level objects will grow as the HPSS architecture is used to accommodate applications that may not be file-based, such as digital libraries, object stores, and large data management systems.

3. HPSS SOFTWARE ARCHITECTURE AND INFRASTRUCTURE

A simplified view of major HPSS software components is shown in Figure 2. Servers are shown together with their basic communication paths (thin lines). The thicker lines show data movement. Infrastructure components (the *glue* holding servers together) are shown at the top. Where multiple boxes of a particular server appear, it indicates that more than one of those servers may be running in a specific site implementation.

3.1 Servers

The *Name Server* maps a file name to an HPSS *bitfile* object. This Name Server provides a POSIX view of a hierarchical name space structure consisting of directories, files, and links. File names are human readable ASCII strings. In addition to mapping names to objects, the Name Server provides access verification to objects.

The *Bitfile Server* provides an abstraction of logical bitfiles to its clients. A logical bitfile is an uninterpreted bit string and is identified by a *bitfile id*. Mapping of a human readable name to the bitfile id is provided by the Name Server. Clients may reference byte-addressable portions of a bitfile by specifying the bitfile id, a starting address, and length. Using one or more Storage Servers, the Bitfile Server maps logical portions of bitfiles onto physical storage devices using *storage segments*. COS is primarily used to support this mapping of logical to physical storage and thus assist the Bitfile Server in choosing appropriate physical storage.

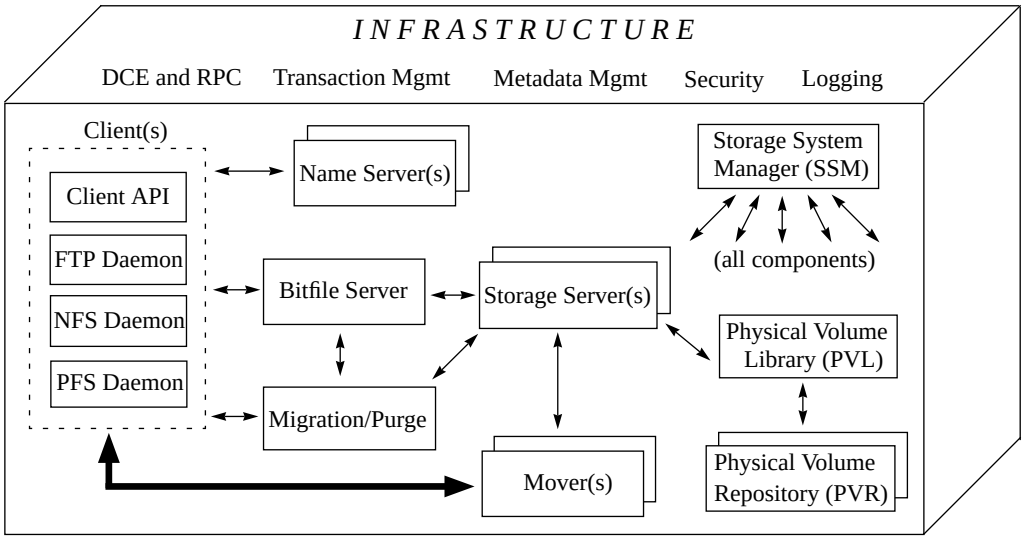


Figure 2. HPSS Software Architecture Diagram

The *Storage Server* provides a three-layer hierarchy of storage objects: storage segments, *virtual volumes* and *physical volumes*. All layers of the Storage Server can be accessed by its clients. The Storage Server translates references to storage segments into references to virtual volumes and finally to physical volumes. It also schedules the mounting and dismounting of removable media. Clients of the Storage Server are typically the Bitfile Server at the segment interface and the Storage System Manager at the virtual and physical volume interface.

The *Mover* is responsible for transferring data from a *source* device to a *sink* device. A device can be a standard I/O device with geometry (e.g., a tape or disk), or a device without geometry (e.g., a network or memory). The Mover also performs a set of device control operations.

The *Physical Volume Library* (PVL) manages all HPSS physical volumes. Clients can ask the PVL to atomically mount and dismount sets of physical volumes. Clients can also query status and characteristics of physical volumes. The PVL maintains mappings of physical volumes to *cartridges*, and cartridges to PVRs. The PVL also controls allocation of drives. When the PVL accepts client requests for volume mounts, the PVL allocates resources to satisfy the request.

The *Physical Volume Repository* (PVR) manages HPSS cartridges. Clients ask the PVR to

mount, dismount, inject and eject cartridges. Every cartridge in HPSS must be managed by exactly one PVR. Clients can also query the status and characteristics of cartridges.

The *Migration and Purge Server* provides storage management facilities for HPSS. This server moves (or copies) bitfiles (or storage segments) from one storage level down to the next as specified in a hierarchy data structure to allow space on the original level to become free. Disk migration is used to free disk space. Tape migration is used to free tape volumes.

The *Storage System Manager* (SSM) monitors and controls resources of the storage system according to site policies. Monitoring includes querying values of managed object attributes representing storage system resources, and receiving notification of fault alarms and significant events. Resource control includes abilities to set managed object attribute values and storage system policy parameters. SSM may also request specific operations be performed on resources within the system (e.g., adding and deleting logical or physical resources). HPSS managed objects are based on OSI management model concepts.

3.2 Infrastructure

HPSS design uses a DCE service infrastructure, including DCE RPCs for control messages and DCE threads for multitasking. HPSS uses DCE Security, Cell Directory, and Time services as well. A library of DCE convenience functions was also developed for HPSS to facilitate server communication and to detect failing components.

Requests to HPSS to perform actions, such as creating bitfiles or accessing data, result in client-server interactions between multiple HPSS components. Transactional integrity to guarantee consistency of server state and metadata is required if a component should fail. Encina, a Transarc product, was selected by the HPSS project as its transaction manager and provides distributed commit-abort semantics, transactional RPCs, and nested transactions. Each HPSS software component has metadata associated with the objects it manages, and each server requires an ability to reliably store its metadata. The Structured File Server, another Encina product, is used by HPSS as a metadata manager and is integrated with the transaction manager.

The security components of HPSS provide authentication, authorization, enforcement, and audit capabilities for the HPSS components. HPSS developed security libraries that utilize DCE security. The authentication service, which is part of DCE, is based on Kerberos v5. A logging service records alarms, events, requests, security audit records, accounting records, and trace information from system components. A central log and local-node logs are supported. A *delog* function is provided to extract, format, and display log records. Delog options support filtering by time interval, record type, server, and user.

3.3 Interfaces

HPSS provides several data transfer interfaces. The Client API provides an interface that mirrors POSIX.1 specifications. Extensions to the POSIX interface are also provided to utilize HPSS parallel data transfer capabilities, and to allow applications to take advantage of COS hint and priority structures that can be passed during file creation.

HPSS also provides standard and parallel FTP server interfaces to transfer files from HPSS to a local file system. Parallel FTP, an extension of standard FTP, was implemented to provide high performance data transfers and provides high performance FTP transfers to the client while still supporting standard FTP commands. Use of Parallel FTP requires additional FTP client code.

The NFS Server interface provides transparent access to HPSS name space objects and bitfile data for client systems through the industry-standard Network File System interface. HPSS also can act as an external file system to the IBM SPx Parallel File System (PFS). The user may issue

a command from an application to import or export files directly to or from HPSS Movers to PFS. COS specifications may be provided in the PFS import/export request to HPSS to facilitate parallel data transfers between systems.

4. HPSS USE OF COS

COS in HPSS defines a set of performance, media, and usage attributes related to the behaviors of a bitfile and its underlying physical storage. Every bitfile must have a COS identifier associated with it. The attributes of a COS are implicitly or explicitly linked with one or more *device hierarchies* and *storage classes* within the storage system. Device hierarchies in HPSS represent particular combinations of storage devices with policies controlling caching and migration of data between the devices. A storage class identifies the storage *type* (e.g., disk, tape) of a particular device, together with COS-related characteristics of the device. COS definitions, associated hierarchy identifiers, and storage classes are used by the Bitfile Server to select appropriate devices and servers for space allocation and new storage segment creation. Each COS definition used by the Bitfile Server is stored in Encina as non-volatile metadata. A simplified COS metadata structure kept by the Bitfile Server is shown below:

```
struct bfs_cos_md {
      Version;          /* HPSS version number        */
      COSId;            /* Class of Service identifier*/
      OpsSupported;     /* I/O Operations supported   */
      MaxSize;          /* Max size of bitfiles in COS*/
      MinSize;          /* Min size of bitfiles in COS*/
      Activity;         /* Amount of expected access  */
      Reliability;      /* Expected reliability level */
      XferRate;         /* Expected transfer rate     */
      Latency;          /* Expected transfer delay    */
      HierId;           /* Associated hierarchy id    */
      } bfs_cos_md_t;
```

In comparison, a storage class metadata structure in HPSS will contain many device-dependent attributes as shown in the following example:

```
struct hpss_sclass_md {
      SClassId;         /* Storage Class identifier   */
      SClassType;       /* Device type for this class */
      TransferRate;     /* Transfer rate in kilobytes */
      StripeSize;       /* No. of elements in a stripe*/
      StripeWidth;      /* Size of a stripe in bytes  */
      BlockSize;        /* Blocksize used on device   */
      OptimalAccessSize;/* Size for best data transfer*/
      StorageSegmSize;  /* Segment size used by client*/
      MaxFileSize;      /* Max size in bytes for class*/
      MinFileSize;      /* Min size of bytes for class*/
      MigrPolicyId;     /* Migration policy to use    */
      PurgePolicyId;    /* Purge policy to use        */
      MPSId;            /* Migration/Purge Server Id  */
      MediaType;        /* General type (e.g., tape)  */
      MediaSubType;     /* Specific type (e.g., 3490E)*/
      AvgLatency;       /* Delay before start of xfer */
      WriteOps;         /* Valid write I/O operations */
      ReadOps;          /* Valid read I/O operations  */
      } hpss_sclass_md_t;
```

Access activity is typically daily, weekly, monthly, or archival. Latency is the delay in seconds between the time a request is received by a Storage Server and the time data begins to be transmitted. This will normally be non-zero for tape devices due to mount delays. Valid I/O opera-

tions for a storage class may be RANDOM, PARALLEL, WRITE, WRITE_MANY, APPEND, and READ. A COS hints structure and a COS priorities structure, both roughly equivalent to the COS definition described above, also exist to assist a client in selecting a suitable COS definition for a newly created bitfile. The priorities structure allows an HPSS client to specify a weighting for each attribute supplied in the hints structure. Priorities currently represent one of the following values: NONE, LOW, DESIRABLE, HIGHLY_DESIRABLE, or REQUIRED. Pointers to the hints and priorities structures are two of the input parameters to the Bitfile Server *bfs_Create* API, which is used to create a new bitfile, allocate space, and save relevant metadata. A pointer to the COS definition structure *actually* used by the Bitfile Server when creating the new bitfile is returned to the client as an output parameter of the call.

Using the HPSS Client API library, applications can specify an existing COS identifier for a file, or fill in the COS hints and priorities structures to describe desired/required service attributes for the file. The FTP quote command can also be used to specify a COS identifier when using HPSS's FTP Daemon. Creating a new file through the Client API is performed through an *hpss_Open* call whose input parameters include (possibly null) pointers to COS hints and priorities structures. When null pointers are passed, the Bitfile Server is free to use a default COS definition for the new bitfile. The hpss_Open call returns a pointer to the COS definition used by the Bitfile Server. The COSId of a bitfile can be obtained or modified by Bitfile Server *bfs_BitfileGetAttrs* and *bfs_BitfileSetAttrs* calls. Changing the COSId may be subject to administrative or operational constraints. The Bitfile Server also maintains the necessary device hierarchy and storage class information that describe where and how the storage segments that comprise the bitfile were physically stored. Bitfiles may reside on multiple devices simultaneously depending on the migration and caching schemes employed in a specific hierarchy.

In initial versions of HPSS, COS use is preliminary and some attributes supplied in a client-generated hints structure will not affect system behavior. Currently, only the transfer rate attribute has significant effect. The Bitfile Server either uses the specific COS identifier supplied by the client, or finds an *appropriately close* COS identifier based primarily on the supplied transfer rate value. If the client specifies the transfer rate's priority as REQUIRED, and the Bitfile Server does not have an existing COS definition that can satisfy the desired rate, the request fails and a *NO_SUPPORT* error is returned to the client. Similarly, if an invalid COS identifier is requested by a client, an error will be returned. Valid COSs are those that have been previously defined in an SSM administrative procedure to create new Storage Server virtual volumes and storage maps (the entities that actually provide storage space in HPSS). Virtual volumes and storage maps are identified by storage class, and are used to provide storage segments of that class to clients of the Storage Server.

The storage segment service is the mechanism used to obtain and access internal storage resources. Clients of the Storage Server are presented with a storage segment address space from 0 to N-1 where N is the byte length of the segment. The Bitfile Server provides a storage class identifier and an allocation length during creation of new storage segments. To ensure locating free space of appropriate type, the storage class must represent storage service conforming to any client-specified COS hints and priorities. During the creation of new space, only storage maps that have proper storage class are searched. If no storage map exists to fit the requirements, a *NO_SPACE_FOUND* error is returned.

The COS structure was designed to be extensible, and additional attributes are planned to more heavily influence server actions during data placement, data transfer, and file/fragment migration operations. A goal for future releases is better integration with large data management systems, whose needs will require COS attributes for objects other than files. In particular, I/O operations on data fragments necessary for resolving complex database queries will require new

COS capabilities. COS attributes are planned for controlling placement or collocation of related files and data fragments on physical media to enable better use of HPSS by new data management applications.

In the current COS implementation, an administrator must be responsible for creating COS and storage class structures at the time HPSS servers are configured. This is necessary because the storage resource objects managed by the Storage Server (i.e., virtual volumes, storage segments, and storage maps) are all identified by the kind of storage they support. At least one COS must be created for the Bitfile Server to use as a default for client requests that do not specify a COS identifier or COS hint and priority structures. Using HPSS Storage System Management facilities, which are based on an X-Windows graphical user interface environment, administrators create new physical volumes, then virtual volumes, and finally, storage maps for the virtual volumes. These must all exist before the creation of any storage segments. When an administrator creates a COS for the Bitfile Server, an accurate determination must be made whether the attribute combination for the COS is sound. Definition of these structures might be based on *a priori* knowledge of devices. Specifying a COS needing a stripe width of four to meet a high data rate when HPSS has only two drives at its disposal for parallel transfers would not work. Administrative creation, modification, or deletion of metadata representing COS and storage class is accomplished through the SSM management windows.

5. RELATED WORK

In previous hierarchical storage management systems used at Lawrence Livermore National Laboratory (LLNL), QoS and COS capabilities were rare. A storage system called FILEM, in use between 1976 and 1986 at LLNL's National Energy Research Supercomputer Center (NERSC), restricted users to specifying a life-span code of *archival*, *long-life*, or *medium-life*. Archival kept the file forever, but forced its migration to the lowest level device, at that time a manual shelf operation. In exchange for long delays on retrieval, the user was charged a lower cost. Long-life also kept the file in the system indefinitely, but an attempt was made to maintain the file in a robotic archive for faster data retrieval than shelf. Medium-life caused the file to be deleted after a time period determined by local site policies. Medium-life also tended to keep the file on disk for faster access, but at substantially higher cost. No other attributes were available to influence the level of service received or corresponding cost accrued.

The Common File System (CFS) [12], developed by Los Alamos National Laboratory (LANL) in 1980 and still running at LANL, NERSC, and other DOE laboratories, provided additional QoS mechanisms that were somewhat improved but still limited. CFS allowed users to specify a usage characteristic for new files (or to change that attribute for existing ones). Users could tell the storage system that the file was to be active daily, weekly, monthly, infrequently, or for only for a few days and then never again. The system used this access hint to place the file at an appropriate initial level in the storage hierarchy, later migrating it to lower levels accordingly. Users also could specify that a file be written to sets of mutually exclusive devices. If a user wanted to write a file twice, and ensure each copy ended up on separate groups of disks or tapes for the life of the file, it could be done with one command during initial storage.

The ability to determine disposition of data improved with NSL-UniTree, an early software development project of the NSL. In NSL-UniTree, dynamic storage hierarchies [13] were implemented to let clients define into what hierarchy their file would be placed. The placement determined how caching and migration to different storage devices would be performed and affected access time to the file, as well as data rates. Clients of NSL-UniTree were able to specify a hierarchy identifier when creating and storing a new file in the same manner as specifying COS identifiers in HPSS. Dynamically managed hierarchies eased the insertion of new

technologies and allowed existing files to automatically take advantage of new devices.

New kinds of metadata, resource attributes, and other abstractions have been proposed or implemented to help optimize use of archival storage systems by non-traditional clients, including relational or object-oriented databases and scientific data management systems [14,15,16,17]. Non-traditional clients do not necessarily use a file as the data entity that is stored and retrieved. In many of these applications, specifying values for overall data transfer rate or parallel transfer stripe widths, as is done in HPSS, may not be meaningful. Other performance issues such as overall latency reduction, close clustering of related data chunks, deadline and continuity requirements, data compression, and redundancy may take precedence.

A recent IEEE-sponsored effort [18] to investigate metadata issues for access to large scientific and technical databases explored problems of storage and archive. Metadata requirements are driven by applications, but also affect storage and software system performance. Since metadata is used to improve the understanding of data content, but also to describe data access concerns, system-level metadata addressing accessibility is closely related to HPSS COS. Attributes are needed that address files or data fragments related by application usage and how these fragments should be stored and migrated. This becomes important in applications that need to manage and query specific, but possibly widely scattered, pieces of information in large data collections. There are several efforts underway to better understand requirements to effectively manage large volumes of scientific data stored on mass storage devices.

One such project, Optimass [19], has ties to the developers of HPSS, and concentrated on multidimensional climate modelling data. In Optimass, large datasets are passed through a partitioning engine driven by several query prediction tools that help estimate data usage patterns. Data fragments, related by application use, are then stored appropriately in the archival system. Fragments are also re-assembled after retrieval from the storage system, based on actual application queries. In Optimass, the partitioner constructs and stores partitioning information in an external metadata database for subsequent use by the reassembler as necessary. This project designed a COS-like interface between the data partitioning/reassembly engines and high-performance mass storage systems such as NSL-UniTree and HPSS. The interface provides an ability to influence or control allocation of space and physical placement of data by defining several key COS attributes associated with *data clusters* (fragments of data related by application use), and to provide these attributes to the storage system through modified client interfaces. This permits the storage system to intelligently bundle the data clusters for a targeted tertiary storage device (usually a slow, sequential-access tape).

The Sequoia 2000 project [20] and the related Mariposa effort [21] are also investigating integration of storage and large data management systems. These projects are working on extending database management system optimizations to deal effectively with tertiary devices and the movement of data between storage systems. Mariposa proposes using several QoS and Trading Function concepts, including subcontracts between subsystems, and open competition for services in a free-market economic model, to explore service guarantees among distributed, cooperating servers. Some of these servers will be performing hierarchical storage management tasks. A typical application might be to provide guaranteed delivery of frames (at a fixed rate) to a high-resolution rendering engine and display. A storage system may need to decide whether or not to accept or decline a subcontract for data movement out of tertiary storage at a specific transfer rate as part of an overall contract guarantee made by a networking service.

6. CONCLUDING REMARKS

Most current work on QoS concepts for high-performance storage, including HPSS, concentrate on attributes related to the cost and performance concerns of initial hierarchical data place-

ment and subsequent data transfer speeds. Emphasis on QoS attributes for other issues such as guaranteed delivery, reliability, and continuity is also needed. Providing storage systems with negotiating capabilities in free-market network environments may also be required. Understanding guaranteed services and third-party brokers using RM-ODP Trading Function ideas would be a beneficial addition to new mass storage system implementations and could provide better communication with new types of storage service consumers.

A significant problem is that archival mass storage systems and new consumers of storage, such as large data management systems, do not communicate well. This is an ongoing research area, but continues to present problems for non-traditional clients of storage service. For example, storage systems and database systems can both provide various request optimization, data replication, and parallel execution capabilities, but integration of these two types of systems is difficult. An ability for customers to negotiate and receive adequate QoS means high-performance mass storage developers must address how to communicate available services to a consumer-oriented world, possibly through brokers. This involves more than development of standard or extended file transfer interfaces.

Properly applying metadata to manage data storage and access has also not yet been addressed in a systematic manner. An issue for HPSS is how to decide where application-related metadata and COS information belongs. Does this information always belong in an external database? How should the information be translated into HPSS COS attributes? Existing Encina metadata management capabilities in HPSS are not infinitely scalable. HPSS is investigating increasing and decreasing the total metadata associated with storage objects under server control and effects on system efficiency.

As requirements grow for high-performance storage systems to support application-specific views instead of traditional file-system views, the need for a richer set of COS and QoS attributes for storage is obvious. As storage systems and computing environments become more distributed, the need to provide better alignment between high-performance storage and open distributed processing standards is also clear. We have incorporated several open systems concepts into the HPSS hierarchical storage management design, but new extensions to COS are required to become aligned with RM-ODP QoS concepts. We believe this to be worth further investigation.

ACKNOWLEDGMENTS

This work was, in part, performed by the Lawrence Livermore National Laboratory under contract number W-7405-Eng-48 and Cooperative Research and Development Agreements under auspices of the U.S. Department of Energy, and by IBM U.S. Federal under High Performance Data Systems Independent Research and Development and other internal funding. For more information about the National Storage Laboratory and HPSS contact:

Dick Watson, LLNL (or) Bob Coyne, IBM U.S. Federal
+1 510 422 9216 +1 713 282 8039
dwatson@llnl.gov coyne@vnet.ibm.com

Access the HPSS tutorial on World Wide Web at URL http://www.ornl.gov/HPSS/HPSS.html

REFERENCES

1. S. Coleman and S. Miller (eds.), *Mass Storage System Reference Model: Version 4*, IEEE Technical Committee on Mass Storage Systems and Technology, May 1990.

2. R. Garrison, et al. (eds.), *Reference Model for Open Storage Systems Interconnection: Mass Storage Reference Model Version 5*, IEEE Storage System Standards Working Group, September 1994.

3. ISO/IEC JTC1/SC 21/WG7, *The Basic Reference Model of Open Distributed Processing*, ITU-TS Recs. X.901 to X. 904 | ISO/IEC 10746, 1994.

4. R. Coyne, H. Hulen, and R. Watson, The High Performance Storage System, *Proc. Supercomputing '93*, Portland, OR, November 15-19, 1993.

5. D. Teaff, R. Coyne, and R. Watson, The Architecture of the High Performance Storage System, *Fourth NASA GSFC Conference on Mass Storage Systems and Technologies*, College Park, MD, March 1995.

6. R. Coyne, and R. Watson, The National Storage Laboratory: Overview and Status, *Proc. Thirteenth IEEE Symposium on Mass Storage Systems*, Annecy, France, June 13-16, 1994.

7. S. Coleman and R. Watson, The Emerging Paradigm Shift in Storage System Architectures, *Special Issue on Storage, Proc. of the IEEE,* April 1993.

8. S. Coleman and R. Watson, New Architectures to Reduce I/O Bottlenecks in High Performance Systems, *Proc. 26th Hawaii International Conference on System Sciences*, Maui, HI, January 5-8, 1993.

9. S. Howe (ed.), *High Performance Computing and Communications: Toward a National Information Infrastructure*, A Report by the Committee on Physical, Mathematical, and Engineering Sciences; Federal Coordinating Council for Science, Engineering, and Technology; Office of Science and Technology Policy, 1994.

10. ISO/IEC JTC1/SC21/WG7, *Draft ODP Trading Function,* ITU-TS Rec. X.9tr | ISO/IEC 13235, 1994.

11. R. Hyer, R. Ruef, and R. Watson, High Performance Direct Network Data Transfers at the National Storage Laboratory, *Proc.Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 26-29, 1993.

12. T. McLarty, B. Collins and M. Devaney, A Functional View of the Los Alamos Central File System, *Digest of Papers, Sixth IEEE Symposium on Mass Storage Systems*, Vail, CO, June 1984.

13. L. Buck and R. Coyne, Dynamic Hierarchies and Optimization in Distributed Storage Systems, *Digest of Papers, Eleventh IEEE Symposium on Mass Storage Systems*, October 7-10, 1991.

14. L. Roelofs and W. Campbell, Applying Semantic Data Modeling Techniques to Large Mass Storage System Designs, *Digest of Papers, Tenth IEEE Symposium on Mass Storage Systems*, Monterey, CA, May 7-10, 1990.

15. R. Grossman, et al., A Proof-of-Concept Implementation Interfacing an Object Manager with a Hierarchical Storage System, *Proc. Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 26-29, 1993.

16. M. Tankenson and S. Wright, A Data Distribution Strategy for the 90s (Files Are Not Enough), *Compilation of Papers, Third NASA GSFC Conference on Mass Storage Systems and Technologies*, College Park, MD, Oct. 1993.

17. J. Shiers, Data Management Requirements for High Energy Physics in the Year 2000, *Proc. Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 26-29, 1993.

18. S. Louis and M. Gary, Storage and Archive Group Summary Report, IEEE Computer Society Technical Committee on Mass Storage Systems Workshop on Metadata for Scientific and Technical Databases, May 16-18, 1994.

19. L. Chen, et al., Efficient Organization and Access of Multi-Dimensional Datasets on Tertiary Storage Systems, submitted to *Information Systems Journal, Special Issue on Scientific Databases,* to be published 1995.

20. M. Stonebraker, J. Frew and J. Dozier, *The Sequoia 2000 Architecture and Implementation Strategy*, Sequoia 2000 Technical Report 93/23, University of California, Berkeley, March 1993.

21. M. Stonebraker, P. Aoki, R. Devine, W. Litwin and M. Olson, Mariposa: A New Architecture for Distributed Data, *Proc. Tenth Int. Conference on Data Engineering*, Houston, TX, February 1994.

Technical Information Department • **Lawrence Livermore National Laboratory**
University of California • Livermore, California 94551